AR
ZFW

TRANSMITTAL FORM

(To be used for all correspondence after initial filing)

Total Number of Pages in This Submission	40

Application No.	09/992,881
Filing Date	November 5, 2001
First Named Inventor	Christopher B. Wilkerson
Art Unit	2183
Examiner Name	Meonske, Tonia L.
Attorney Docket Number	42390P11933

ENCLOSURES ***(check all that apply)***

- [X] Fee Transmittal Form
  - [ ] Fee Attached
- [ ] Amendment / Response
  - [ ] After Final
  - [ ] Affidavits/declaration(s)
- [ ] Extension of Time Request
- [ ] Express Abandonment Request
- [ ] Information Disclosure Statement
  - [ ] PTO/SB/08
- [ ] Certified Copy of Priority Document(s)
- [ ] Response to Missing Parts/ Incomplete Application
  - [ ] Basic Filing Fee
  - [ ] Declaration/POA
  - [ ] Response to Missing Parts under 37 CFR 1.52 or 1.53
- [ ] Drawing(s)
- [ ] Licensing-related Papers
- [ ] Petition
- [ ] Petition to Convert a Provisional Application
- [ ] Power of Attorney, Revocation Change of Correspondence Address
- [ ] Terminal Disclaimer
- [ ] Request for Refund
- [ ] CD, Number of CD(s)
- [ ] After Allowance Communication to Group
- [ ] Appeal Communication to Board of Appeals and Interferences
- [X] Appeal Communication to Group **(Appeal Notice, Brief, Reply Brief)**
- [ ] Proprietary Information
- [ ] Status Letter
- [X] Other Enclosure(s) *(please identify below):*

Return receipt postcard

Remarks

**Response to Non-Compliant Appeal Brief under C.F.R. Section 41.37

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

Firm or Individual name	Joseph Lutz, Reg. No. 43,765 BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
Signature	
Date	July 12 2005

CERTIFICATE OF MAILING/TRANSMISSION

I hereby certify that this correspondence is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to: Mail Stop Appeal Brief-Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Typed or printed name	Marilyn Bass		
Signature		Date	07-12-05

FEE TRANSMITTAL for FY 2005

Patent fees are subject to annual revision.

☐ Applicant claims small entity status. See 37 CFR 1.27.

TOTAL AMOUNT OF PAYMENT ($)

Complete if Known	
Application Number	09/992,881
Filing Date	November 5, 2001
First Named Inventor	Christopher B. Wilkerson
Examiner Name	Meonske, Tonia L.
Art Unit	2183
Attorney Docket No.	42390P11933

METHOD OF PAYMENT *(check all that apply)*

☐ Check ☐ Credit card ☐ Money Order ☒ None ☐ Other (please identify): _____

☒ Deposit Account Deposit Account Number: 02-2666 Deposit Account Name: Blakely, Sokoloff, Taylor & Zafman LLP

For the above-identified deposit account, the Director is hereby authorized to: (check all that apply)

☐ Charge fee(s) indicated below
☒ Charge any additional fee(s) or underpayment of fee(s) under 37 CFR §§ 1.16, 1.17, 1.18 and 1.20.
☐ Charge fee(s) indicated below, except for the filing fee
☐ Credit any overpayments

FEE CALCULATION

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet.	
2053	130	2053	130	Non-English specification	
1251	120	2251	60	Extension for reply within first month	
1252	450	2252	225	Extension for reply within second month	
1253	1,020	2253	510	Extension for reply within third month	
1254	1,590	2254	795	Extension for reply within fourth month	
1255	2,160	2255	1,080	Extension for reply within fifth month	
1401	500	2401	250	Notice of Appeal	
1402	500	2402	250	Filing a brief in support of an appeal	
1403	1,000	2403	500	Request for oral hearing	
1451	1,510	2451	1,510	Petition to institute a public use proceeding	
1460	130	2460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
1809	790	1809	395	Filing a submission after final rejection (37 CFR § 1.129(a))	
1810	790	2810	395	For each additional invention to be examined (37 CFR § 1.129(b))	

Other fee (specify) _____

SUBTOTAL (2) ($)

SUBMITTED BY Complete (if applicable)

Name (Print/Type)	Joseph Lutz	Registration No. (Attorney/Agent)	43,765	Telephone	(310) 207-3800
Signature				Date	07-12-05

Based on PTO/SB/17 (12-04) as modified by Blakely, Sokoloff, Taylor & Zafman (wlr) 12/15/2004.
SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA ~~22313-1450~~

Our Ref. No.: 42390P11933

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:	Examiner: Meonske, Tonia L.
Christopher B. Wilkerson	Art Unit: 2183
Application No.: 09/992,881	
Filed: November 5, 2001	
For: SYSTEM AND METHOD TO BYPASS EXECUTION OF INSTRUCTIONS INVOLVING UNRELIABLE DATA DURING SPECULATIVE EXECUTION	

RESPONSE TO NON-COMPLIANT APPEAL BRIEF UNDER C.F.R. §41.37

Mail Stop Appeal Brief - Patent
Commissioner for Patents
P. O. 1450
Alexandria, VA 22313-1450

Dear Sir:

In response to the Notification of Non-Compliant Appeal Brief mailed June 22, 2005, Applicant submits the following Appeal Brief pursuant to 37 C.F.R. §41.37(c) for consideration by the Board of Patent Appeals and Interferences. Please charge any additional amount due or credit any overpayment to deposit Account No. 02-2666.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

Christopher B. Wilkerson, the party named in the caption, transferred his rights to that which is disclosed in the subject application through an assignment recorded on November 5, 2001 (012328/0021) to Intel Corporation, of Santa Clara, California. Thus, as the owner at the time the brief is being filed, Intel Corporation, of Santa Clara, California is the real party in interest.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences which will affect or be affected by the outcome of this appeal.

III. STATUS OF CLAIMS

Claims 1-38 are pending. Claims 1-4, 6-15, 17-22, 24-29 and 35-38 are rejected in this application and Claims 5, 16, 23 and 30 are objected to. Claims 31-34 are allowed in this application. Applicant hereby appeal the rejection of rejected Claims 1-4, 6-15, 17-22, 24-29 and 35-38 and the objection of Claims 5, 16, 23 and 30. However, Applicant respectfully submits that allowed Claims 31-34 do not form part of the Appeal.

IV. STATUS OF AMENDMENTS

The claims are amended in accordance with the Response Amendment filed on January 11, 2005, wherein Claims 1 and 31. The claim amendments and new claims requested in the Response Amendment filed on January 11, 2005 regarding amended Claims 1 and 31 were entered.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The pending claims relate to a system and method to bypass execution of instructions involving unreliable data during speculative execution. Claims 1, 19 and 26 recite analogous claim features. Claim 19 is representative.

Claim 19 recites features including:

> a memory, a storage device, and a processor each coupled to a bus;
> the processor including an execution engine having instructions which when executed by the processor cause the processor to perform actions including:
> identifying scratch values generated during speculative execution of a processor;
> setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and
> bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value. (Emphasis added.)

As illustrated in FIG. 1B of Applicant's specification, processor 110 includes an execution engine 160. As indicated by Applicant's specification:

instructions and data dependent on invalid data are referred to as scratch values (SVals). Other data values not derived from SVals are referred to as reliable values (RVals). (pg. 4, lines 17-20.)

As also indicated by Applicant's specification:

the execution of instructions dependent on SVals is ineffective and may be counterproductive. One study has shown that forty percent (40%) of all instructions executed during scratch mode are dependent on SVals and can be completely ignored. (pg. 4, lines 19-22.)

As further indicated by Applicant's specification:

Each data storage location in a processor may be augmented with an SVal tag in the form of a single bit. In this embodiment, each register, predicate, flag, etc. may have an additional bit appended called an SVal tag bit. (pg. 5, lines 30-33.)

As further described by Applicant's specification:

To take advantage of the addition of SVal tags within a processor, a processor's internal execution engine 160 may be written to implement the method described herein. (pg. 7, lines 4-6.)

The methods described herein involve handling instructions dependent on SVals and instructions dependent on RVals differently. For, example, FIG. 3 of Applicant's specification illustrates a flowchart, which includes a flow of actions taken when processing an instruction and its operands by the execution engine within a processor. (pg. 8, lines 17-20.) As illustrated in FIG. 3:

A current instruction is obtained, as shown in block 310. In one embodiment, when a new instruction is fetched into the pipeline, the instruction's SVal tag is initially set to zero (0). The operands of the instruction are then examined for the presence of SVals, as shown in block 320. . . . If at least one operand of the current instruction is an SVal, then the SVal tag is propagated to the destination, as shown in block 324, such that the SVal tag of the destination is set to one (1). If the SVal tag of at least one of the instruction's operands is an SVal, that is it is set to one (1), the SVal tag associated with the instruction is set to one (1), as shown in block 330. Execution of the instruction is bypassed if at least one operand of the instruction has an SVal set to one, as shown in block 340. A check may be made to determine whether the instruction has been bypassed, as shown in block 350. If the instruction has not been bypassed, the instruction is executed, as shown in block 360. If the execution of the instruction has been bypassed, the next instruction is obtained, as shown in block 370, and the flow of actions continues at block 320. Similarly, after the current instruction is executed, as shown in block 360, the next instruction is obtained, as shown in block 370, and the flow of actions continues at block 320. When an instruction executes, the instruction's SVal tag value is copied to the destination register(s). (pg. 8, lines 20 - pg. 9, line 7.)

Based on the cited passages above, the scratch values generated during speculative execution of a processor are identified using the SVal tag associated with the operands of a processor. Using this information, an execution engine of a processor may bypass execution of instructions having at least one operand with an associated tag that indicates that the operand is a

scratch value generated during speculative execution of a processor, as recited by Claims 1, 19 and 26.

As illustrated in FIG. 1 of Applicant's specification, processor 110 includes an execution engine 160. Claim 8 recites a processor 110 comprising

> a plurality of registers 130 having a corresponding plurality of register tags 134 to indicate whether the data 132 stored in the register holds a scratch value;
> a plurality of flags 140 having a corresponding plurality of flag tags 142 to indicate whether the data 141 reflected by the flag 140 is based on a scratch value;
> a plurality of predicates 144 having a corresponding plurality of predicate tags 146 to indicate whether the data 145 reflected by the predicate 144 is based on a scratch value; and
> an execution engine 160 to bypass execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value.

Accordingly, as recited by Claim 8, an execution engine 110 may bypass execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value.

Claim 12 recites a processor comprising at least two arithmetic units, a translation look aside buffer, a branch prediction unit and an execution engine, for example, as illustrated with reference to FIG. 1B of Applicant's specification. The execution engine recited by Claim 12 may include a plurality of instructions, which when executed, cause the processor to perform actions including:

> identifying scratch values generated during speculative execution of a processor,
> setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and
> bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value. (Emphasis added.)

Accordingly, as recited by Claim 12, the execution engine may bypass execution of instructions during speculative execution of the processor if the instruction has an operand with an associated tag that indicates that the operand is a scratch value.

Claims 7, 18 and 25 recite:

> propagating the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value.

As indicated by Applicant's specification,

> In one embodiment, store instructions are sorted into two classes before being permitted to write to the store buffer. The first class of store instructions includes store instructions whose address generation register operands are scratch values. The second class of store instructions includes store instructions marked as scratch values whose address generation register operands are not scratch values. As to the first class, because the destination address is not known, the SVal tag is not propagated. As to the second class, because the destination address is known, the SVal tag is propagated to the store buffer. (pg. 9, lines 19-26.)

Hence, Claims 7, 18 and 25 relate to the second class of instruction, for which, the SVal tag is propagated to the store buffer.

Claim 35 recites:

> A method comprising:
> setting a tag of a register when an instruction having the register as a destination results in a cache miss, to identify the register as containing a scratch value;
> bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value until at least one instruction is detected that results in a cache miss; and
> re-executing bypassed instructions once a cache is loaded with cache miss data. (Emphasis added.)

As indicated by Applicant's specification

> When an instruction is a load and it misses the cache, its destination register SVal tag is set to one (1). When an instruction is a load and it hits the cache but the SVal tag of one or more of its source operands is set to one (1), the SVal tag of its destination register is set to one (1). The SVal tag of an instruction with an immediately available operand is set to zero (0). (pg. 9, lines 9-13.)

Accordingly, in one embodiment, as recited by Claim 35:

> a processor is enabled to selectively quash instructions . . . involving data derived from a cache miss. The results are that power consumption of a processor is reduced, execution resources of the processor are conserved, and throughput of the processor is increased. (pg. 3, lines 16-20.)

Accordingly, the bypassing of execution of instructions having an operand that is a scratch value, as recited by Claim 25, reduces the amount of instructions executed during speculative execution by a processor.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The grounds of rejection involved in this appeal are as follows:

Are Claims 1-4, 6-11, 19-22 and 24-29 unpatentable under 35 U.S.C. §102(b) as being anticipated by U.S. Patent No. 5,799,179 to Ebcioglu et al. ("Ebcioglu")?

Are Claims 35-38 unpatentable under 35 U.S.C. §102(b) as being clearly anticipated by U.S. Patent No. 5,933,627 to Parady ("Parady")?

Are Claims 12-15, 17 and 18 unpatentable under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,651,125 issued to Witt et al. ("Witt") in view of Ebcioglu?

The Examiner has allowed Claims 31-34 over the prior art. Hence, Applicant respectfully submits that allowed Claims 31-34, do not form part of the Appeal.

VII. ARGUMENT

A. Overview of the Cited References

1. Overview of Ebcioglu Reference

The teachings of Ebcioglu are directed to reducing overhead required to resolve exceptions caused by speculative instructions. As described by Ebcioglu:

> According to the present invention, CPU overhead is minimized through: tracking speculative exceptions, for later processing during resolution, including pointing to the addresses of these speculative instructions; and b) resolving these exceptions by correcting the cause of the exception and re-executing the instruction(s) which are known to be in the taken path. (col. 3, line 62 - col. 4, line 1.) (Emphasis added.)

To achieve the minimization of CPU overhead, Ebcioglu explicitly requires that:

> minimization of CPU overhead from exceptions from speculative instructions as well as efficient handling of an exception for a speculative instruction which turns out to be in the taken path. In accordance with the teaching of this invention, an exception from a speculative instruction is processed if and only if it would occur in the original sequential program. (col. 3, lines 31-37.) (Emphasis added.)

To limit the above-described processing of exceptions caused by speculative instructions ("speculative exceptions"), Ebcioglu describes techniques for processing and tracking original speculative exceptions and secondary speculative exceptions. To track such information, Ebcioglu describes populating of target registers, exception bits and exception files, which according to Ebcioglu:

> allows tracking back to the speculative instruction causing the original speculative exception for its resolution. (col. 4, lines 22-24.)

As further described by Ebcioglu, the tracking of speculative exceptions is used to determine instructions which are dependent from the initial instruction that caused the exception. Such information is needed to resolve the speculative exception

As taught by Ebcioglu:

> Exception tracking also discards information for speculative exceptions which turn out to be outside the taken path. Speculative exception resolution is triggered when a non-speculative instruction, which is in the taken path, uses an operand from a register having its exception bit set. Speculative exception resolution includes correcting the exception condition which caused the exception and re-executing the instructions which depend on the results of the instruction causing the speculative exception.
>
> The advantage to this approach is that rather than re-executing all speculative instructions, as does the prior art, only those which depend a) on the instruction causing the exception, and b) are in the taken path, are re-executed. This approach saves CPU processing time. (col. 4, lines 24-37.) (Emphasis added.)

As indicated by the cited passage above, information regarding speculative exceptions is required to identify speculative instructions which depend on the instruction that caused the exception. The additional requirement for processing such speculative instructions is that such instructions are in the taken path. Hence, Ebcioglu teaches that CPU overhead is reduced by limiting re-execution of speculative instructions to those speculative instructions which depend on the instruction causing the exception and are in the taken path. In addition, resources are further reduced by discarding information for speculative exceptions, which turn out to be outside the taken path. (*See*, col. 4, lines 24-25.)

Hence, the teachings of Ebcioglu are specifically limited to processing exceptions caused by a speculative instruction only when the exception would have been caused during the original sequential program (*See*, col. 3, lines 31-37) and limiting the re-execution of speculative instructions to those which depend from the instruction that caused the exception and are in the taken path (*See*, col. 4, lines 36–37.)

As defined by Ebcioglu:

> Speculative instructions have an extra bit, referred to as a speculative bit, which is set. A speculative instruction is an instruction which is moved above a conditional jump which determines whether or not the speculative instruction is in the taken path. Non-speculative instructions have the speculative bit reset. (col. 3, lines 40-45.) (Emphasis added.)

Based on the cited passages above, the teachings of Ebcioglu are not directed to limiting the number of speculative instructions executed during speculative execution, but are, in fact, directed to limiting the amount of speculative instructions, which must be re-executed during speculative exception resolution. (*See*, col. 4, lines 35–37.)

2. Overview of Parady Reference

Parady describes a thread switch on a block load or store using an instruction thread field. Parady is directed to a method and apparatus for switching between threads of a program in response to a long latency event. As described with Parady:

> The indication that a thread switch is required is provided on a line 114 providing an L2-miss indication from cache control/system interface 22 of FIG. 1. Upon such an indication, a switch to the next thread will be performed, using, in one embodiment, the next thread pointer on line 116. The next thread pointer is two bits indicating the next thread from an instruction which caused the cache miss. (col. 3, lines 58-65.) (Emphasis added.)

FIG. 2 of Parady illustrates L2 cache tags, which as known to those skilled in the art, are used to determine whether requested data is contained within an L2 cache. The only reference to element 80 shown in FIG. 1 of Parady is the description that L2 cache tag memory 80, and L2 cache data memory 82 are shown in FIG. 2. (*See*, col. 3, lines 27-28.)

As further described by Parady:

> These two bits of the next thread pointer come from a thread field 118 in an instruction 120. Instruction 120 also includes an opcode field 122 and a source and destination register fields 124 and 126, respectively. By adding the two bit thread field 118 to appropriate instructions, control can be maintained over thread switching operations. In one embodiment, the thread field is added to all load and store operations. (col. 3, line 66 - col. 4, line 6.) (Emphasis added.)

Accordingly, as illustrated in FIG. 3 of Parady, thread switching logic 112, in response to an L2 cache miss received over line 114, switches execution to a next thread according to next thread pointer received over line 116. Hence, Parady teaches that execution of a thread is paused in response to a long latency event. While the thread is paused, execution of a next thread is indicated.

Parady teaches that upon indication of a cache miss, a switch to the next thread is performed using a next thread pointer received on line 16, which may be part of a thread field 118 and an instruction 120, as shown in FIG. 4 of Parady. However, as recited by Parady:

> In a preferred embodiment, upon completion of the memory access which caused the thread switch, the thread with the memory access must wait until it is pointed to again by the round robin or thread pointer bits to continue with its operation. Alternatively, a particular thread could be identified as a critical thread, and generate an interrupt as soon as the memory access is completed. (col. 4, lines 43-48.) (Emphasis added.)

Accordingly, based on the cited passage above, Applicant respectfully submits that the switching between threads in response to long latency events does not cause instructions to be bypassed since execution of such instructions will resume once the thread is pointed to again by the round robin, or thread pointer bits, to continue with its operation. (*See*, col. 4, lines 43-46.)

3. Overview of Witt Reference

Witt describes a high-performance, superscalar microprocessor including a common reorder buffer and common register file for both integer and floating point operations. Hence, Witt teaches a superscalar microprocessor that:

> achieves increased performance without increasing guide size by sharing several key components. The architecture of the microprocessor provides that the integer unit 215 and the floating point unit 225 are coupled to a common data processing bus 535. Data processing bus 535 is a high-speed, high-performance bus primarily due to its wide bandwidth. Increased utilization of both the integer functional unit and the floating point functional unit is thus made possible as compared to designs where these functional units reside on separate buses. (col. 10, lines 15-24.)

To further provide increased performance, Witt teaches:

> The integer and floating point functional units also share a common branch unit 520 on data processing bus 535. Moreover, the integer and floating point

functional units share a common load/store unit 530, which is coupled to the same data processing bus 535. The disclosed microarchitecture advantageously increases performance while more efficiently using the size of the processor die. (col. 10, lines 29-36.)

As described by Witt:

The reorder buffers contain the speculative state of the microprocessor, whereas the register files contain the architectural state of the microprocessor. (col. 3, lines 63-65.)

As further described by Witt:

A program counter (PC) is employed to keep track of the point in the program instruction stream which is the boundary between those instructions which have been retired into register file 235 as being no longer speculative, and those instructions which have been speculatively executed and has resulted in a resident and reorder buffer (ROB) 240 pending retirement. (col. 12, lines 29-36.)

As taught by Witt:

Reorder buffer 240 is a circular buffer or queue which receives out-of-order functional unit results and which updates register file 235 in sequential instruction program order. (col. 11, lines 61-64.)

The reorder buffer is further described by Witt as follows:

Reorder buffer 240 contains positions allocated to renamed registers, and holds results of instructions which are speculatively executed. (col. 12, lines 1-3.)

Witt provides no teachings regarding speculative instructions. Instead, Witt teaches speculative execution:

Instructions are speculatively executed when branch logic predicts that a certain branch will be taken such that instructions in the predicted branch are executed on speculation that the branch was indeed properly taken in a particular instance. If it should be determined that the branch was mispredicted, then the branch results which are in the reorder buffer are effectively cancelled. This is accomplished by microprocessor effectively backing up to the mispredicted branch instruction, flushing the speculative state of the microprocessor and resuming execution from a point in the program instruction stream prior to the mispredicted branch. (col. 12, lines 3-14.)

Witt describes exception handling. As taught by Witt:

When an exception occurs in the execution of a particular instruction, no global action is required. Rather, the exception status is merely reflected and the results status returned to ROB 240. (col. 29, lines 62-65.) (Emphasis added.)

As further described by Witt:

When ROB 240 encountered the exception status in the process of retiring instructions, it initiates a trap operation, which consists of clearing all entries from ROB 240, asserting an exception signal to all functional units to clear them (and IDECODE), generating a trap vector per the VF bit and redirecting the fetcher 257 to trap handling code. The VF bit indicates whether a trap should be taken as an external fetch (as a load from a table of vectors) or internally generated by

concatenating a constant with the vector number. The VF bit is a feature of the architecture of the advanced micro devices AM 29000 microprocessor series. (col. 30, lines 6-16.)

As further described by Witt:

All reservation station entries can be de-allocated within a clock cycle should an exception occur. If a branch misprediction occurs, immediate results are flushed out of the functional units and are de-allocated from the re-order buffer. (col. 38, lines 31-34.)

Hence, Witt fails to provide any teaching or suggestion regarding distinguishing between exceptions caused by speculative instruction and non-speculative instructions.

B. Rejection of Claims 1-4, 6, 19-22, 24 and 26-29 as Anticipated by Ebcioglu

The Examiner rejected pending Claims 1-4, 6, 19-22 24 and 26-29 under 35 U.S.C. §102(b) as being unpatentable over Ebcioglu.

1. Errors of Law and Fact in the Rejection

Applicant respectfully asserts that the Examiner has failed to adequately set forth a *prima facie* rejection under 35 U.S.C. §102(b). "Anticipation requires the presence in a single prior art reference disclosure of each and every element of the claimed invention, *arranged as in the claim.*" *Lindemann Maschinenfabrik v. American Hoist & Derrick* ("Lindemann"), *730 F.2d 452, 1458 (Fed. Cir. 1994)*(emphasis added). Additionally, each and every element of the claim must be exactly disclosed in the anticipatory reference. *Titanium Metals Corp. of American v. Banner* ("Banner Titanium"), *778 F.2d 775, 777 (Fed. Cir. 1985).*

Although the Examiner has rejected Claims 1-4, 6-11, 19-22 and 24-29 as anticipated by Ebcioglu, the Examiner fails to show that each and every element of the claimed subject matter is disclosed in Ebcioglu, as required to establish a *prima facie* case of anticipation under §102(b). Id.

According to the Examiner, Ebcioglu (col. 4, lines 22-26; col. 7, lines 13-16; col. 9, lines 51-63) teaches the bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value, as recited by the claimed subject matter. (*See*, pg. 2, item 3c of Final Office Action mailed November 5, 2004.) However, after careful review of the relevant portions of Ebcioglu cited by the Examiner, Applicant respectfully disagrees with the Examiner's contention. According to the Examiner:

Speculative instructions that are outside of the taken path are discarded or bypassed because they are no longer needed. (*See*, supra.)

Applicant respectfully submits that the Examiner has improperly equated the disregarding of information for speculative exceptions, which turn out to be outside the taken path,

as taught by Ebcioglu (*See*, col. 4, lines 24-26) with the bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value, as recited by the claimed subject matter. Furthermore, the Examiner has incorrectly equated actions performed during speculative exception resolution, as taught by Ebcioglu, with speculative execution of a processor, as recited by the claimed subject matter.

Ebcioglu describes techniques for minimizing CPU overhead required to process exceptions caused by speculative instructions, which are referred to by Ebcioglu as "speculative exceptions." (*See*, col. 3, lines 31-37.) Ebcioglu describes techniques for processing and tracking information regarding speculative exceptions, which include original speculative exceptions and secondary speculative exceptions. (*See*, col. 4, lines 4-24.) To track such information, Ebcioglu describes populating of target registers, exception bits and exception files, which according to Ebcioglu:

> allows tracking back to the speculative instruction causing the original speculative exception for its resolution. (col. 4, lines 22-24.) (Emphasis added.)

As explicitly recited by Ebcioglu:

> Exception tracking also discards information for speculative exceptions which turn out to be outside the taken path. (col. 4, lines 24-26.) (Emphasis added.)

Applicant respectfully submits that the discarding of exception tracking information for speculative exceptions that are outside the taken path does not teach the bypassing of instructions during speculative execution, as recited by the claimed subject matter.

Furthermore, as recited in the Background of Ebcioglu, handling of exceptions caused by speculative instructions requires re-execution of all speculative instructions. (*See*, col. 2, lines 1-6.) As recited by Ebcioglu:

> Eventually, the instruction causing the exception will be re-executed, at which time the exception will be handled. (col. 2, lines 6-7.) (Emphasis added.)

Regarding the re-execution of speculative instructions in response to resolution of a speculative exception, Ebciuglu teaches that:

> Rather than re-executing all speculative instructions, as does the prior art, only those which depend a) on the instruction causing the exception, and b) on the taken path, are re-executed. This approach saves CPU processing time. (col. 4, lines 33-37.) (Emphasis added.)

As defined by Ebcioglu:

> A speculative instruction is an instruction moved above a conditional jump which determines whether or not the speculative instruction is in the taken path. (col. 4, lines 42-45.)

Applicant respectfully submits determining whether a speculative instruction is in the taken path would require waiting for the resolution of the conditional jump, as resolution of the conditional jump would be the only way to determine whether the speculative instruction was actually in the taken path. Applicant submits that once it is determined whether a speculative instruction is in the taken path, the speculative instruction, by definition, is no longer speculative and is, in fact, now a non-speculative instruction (an instruction which would occur in the original sequential program). In other words, instructions which are in the taken path will occur in the original sequential program and hence, are no longer speculative.

Conversely, as indicated by the claimed subject matter, scratch values generated during speculative execution of a processor determine whether an instruction is bypassed during speculative execution of a processor. Hence, Applicant submits that the Examiner has incorrectly equated speculative execution of a processor with actions performed during resolution of speculative exceptions, as taught by Ebcioglu.

Accordingly, Applicant respectfully submits that the entire specification of Ebcioglu, is restricted to teaching the limiting of re-execution of speculative instructions during speculative exception resolution to such speculative instructions, which depend on the instruction causing the exception, and are in the taken path. Applicant respectfully submits that such teachings do not disclose bypassing execution of instructions according to scratch values generated during speculative execution of a processor.

Hence, Applicant submits that the entire description of Ebcioglu is devoid of any reference to bypassing speculative instructions during speculative execution. The case law is quite clear in establishing that each and every element of the claim must be exactly disclosed in the anticipatory reference. Banner Titanium, *Id.* Hence, a *prima facie* case of anticipation of the claims over Ebcioglu has not been established and the rejection of Claims 1-4, 6, 19-22, 24 and 26-29 is therefore erroneous.

2. Specific Limitations Not Described in the Prior Art

Claims 1, 19 and 26 recite analogous claim features. Claim 19 is representative. Claim 19 recites features, including:

> an execution engine having instructions which when executed by the processor cause the processor to perform actions including:
>
> identifying scratch values generated during speculative execution of a processor;
>
> setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and
>
> bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value. (Emphasis added.)

3. Explanation Why Such Limitations Render the Claims Non-Obvious Over the Prior Art

Applicant claims a system and method to bypass execution of instructions involving unreliable data during speculative execution. As indicated at pg. 4, ¶0016 of Applicant's specification, instructions and data dependent on invalid data are referred to as scratch values (SVals). Other data values not derived from SVals are referred to as reliable values (RVals). As further indicated by Applicant's specification:

> Each data storage location in a processor may be augmented with an SVal tag in the form of a single bit. In this embodiment, each register, predicate, flag, etc. may have an additional bit appended called an SVal tag bit. (*See*, pg. 5, ¶0020, of Applicant's specification.)

As further described by Applicant's specification:

> To take advantage of the addition of SVal tags within a processor, a processor's internal execution engine 160 may be written to implement the methods described herein. (*See*, pg. 7, ¶0022, of Applicant's specification.)

In one embodiment, the method described recites the use of scratch values generated during speculative execution of a processor. As recited by Claims 1, 19 and 26, execution of an instruction is bypassed if an operand of the instruction contains a scratch value.

Conversely, the teachings of Ebcioglu are specifically limited to minimizing CPU overhead required to process exceptions caused by speculative instructions, which are referred to by Ebcioglu as "speculative exceptions." (*See*, col. 3, lines 31-37.) To limit the above-described processing of speculative exceptions, Ebcioglu describes techniques for processing and tracking speculative exceptions. To track such information, Ebcioglu describes populating of target registers, exception bits and exception files, which according to Ebcioglu:

> allows tracking back to the speculative instruction, causing the original speculative exception for its resolution. (col. 4, lines 22-24.)

Accordingly, to limit CPU overhead, Ebcioglu teaches that resolution of speculative exceptions is limited to speculative exceptions, which are within the taken path (occur in the original sequential program). As explicitly stated by Ebcioglu:

> In accordance with the teaching of this invention, an exception from a speculative instruction is processed if and only if it would occur in the original sequential program. (col. 3, lines 31-37.) (Emphasis added.)

To further reduce CPU overhead, Ebcioglu teaches:

> Rather than re-executing all speculative instructions as does the prior art, only those which depend a) on the instruction causing the exception, and b) are in the taken path, are re-executed. This approach saves CPU processing time. (col. 4, lines 33-37.) (Emphasis added.)

According to the Examiner:

> Speculative instructions that are outside of the taken path are discarded, or bypassed, because they are no longer needed. (*See*, pg. 2, item 3c, of the Final Office mailed November 5, 2004.)

Applicant respectfully submits that the speculative instructions referred to by the Examiner have already been speculatively executed. However, as taught by Ebcioglu during the initial execution of a speculative instruction, the occurrence of an original speculative exception or a secondary speculative exception results in the storage of information to allow:

> tracing back to the speculative instruction causing the original speculative exception for its resolution. (col. 4, lines 22-24.)

As explicitly taught by Ebcioglu:

> Speculative exception resolution is triggered when a non-speculative instruction, which is in the taken path, uses an operand from a register having its exception bit set. (col. 4, lines 26-29.) (Emphasis added.)

Accordingly, based on the cited passages above, execution of speculative instructions continues until speculative exception resolution is triggered, as described above. However, once speculative exception resolution is triggered, a speculative exception is processed if and only if it would occur in the original sequential program. (*See*, col. 3, lines 31-37.)

Furthermore, rather than re-executing all speculative instructions during speculative exception resolution, as does the prior art, only those which depend on the instruction causing the exception and are in the taken path are re-executed. Hence, Applicant submits that the teachings of Ebcioglu are strictly limited to restricting re-execution of previously executed speculative instructions to speculative instructions which depend on an instruction that caused the exception and are in the taken path. Otherwise, the speculative instruction is not re-executed during speculative exception resolution, as taught by Ebcioglu.

Conversely, the claimed subject matter recites bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value. This feature does not recite bypassing instructions for re-execution, but in fact, recites bypassing execution of instructions during speculative execution of a processor.

Assuming, arguendo, that Ebcioglu taught the bypassing of execution of instructions, Ebcioglu fails to disclose that the bypassing of execution of instructions is based on whether an operand of an instruction contains a scratch value generated during speculative execution of a processor, as indicated by an associated tag of the operand, as recited by independent Claims 1, 19 and 26.

In contrast to the claimed subject matter, the teachings of Ebcioglu are limited to two courses of action when an instruction attempts to use a register having its exception bit set. If the instruction is a speculative instruction, an attempt to use a register having a set exception bit results in a secondary speculative exception. (*See*, col. 4, lines 13-15.) The second course of action occurs

when a non-speculative instruction uses an operand from a register having its exception bit set. Under such a condition, a speculative exception resolution is triggered if the non-speculative instruction is in the taken path. (*See*, col. 4, lines 26-29.)

Accordingly, as taught by Ebcioglu, when an instruction uses an operand from register having its exception bit set, either a secondary speculative exception is created or a speculative exception resolution is triggered. Under either course of action taught by Ebcioglu, an instruction which uses an operand having its exception bit set does not determine whether the instruction is bypassed, as recited by the Claims 1, 19 and 26.

Furthermore, the setting of an exception bit, as taught by Ebcioglu, is not used to determine whether an operand contains a scratch value. As taught by Ebcioglu, the setting of the exception bit of a register, in response to either an original speculative exception or a secondary speculative exception (*See*, col. 4, lines 13-16 and 26-29).

> allows tracking back to the speculative instruction causing the original speculative exception for its resolution. (col. 4, lines 22-24.) (Emphasis added.)

In fact, Applicant respectfully submits that the bypassing of instructions that use an operand from a register having its exception bit set is directly contrary to the teachings of Ebcioglu. Applicant submits that the claimed subject is directly contrary to the teachings of Ebcioglu since Ebcioglu processes such instructions to determine whether speculative exception resolution is triggered or whether a secondary speculative exception is created. Hence, Applicant respectfully submits that the entire specification of Ebcioglu fails to disclose the bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value generated during speculative execution of a processor, as recited by independent Claims 1, 19 and 26.

Yet, in spite of the lack of any teaching toward the bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value generated during speculative execution of a processor of the claimed subject matter, the Examiner incorrectly finds that Ebcioglu discloses each and every element of independent Claims 1, 19 and 26. However, the case law is quite clear in establishing that each and every element of the claim must be exactly disclosed in the anticipatory reference. Banner Titanium, Id. Hence, a *prima facie* case of anticipation of Claims 1, 19 and 26 cannot be established with Ebcioglu as an anticipatory reference. Id.

Therefore, a *prima facie* case of anticipation of the claims is not established and the rejection of Claims 1-4, 6, 7, 19-22 and 24-29 is erroneous and should be overturned. Accordingly, Applicant respectfully requests that the §102(b) rejection of the Claims 1-4, 6, 19–22, 24 and 26-29 be overturned.

C. Rejection of Claims 8-11 as Anticipated by Ebcioglu

The Examiner rejected pending Claims 8-11 under 35 U.S.C. §102(b) as being unpatentable over Ebcioglu.

1. Errors of Law and Fact in the Rejection

The Examiner has made the same errors as previously described with respect to the rejection of Claims 1-4, 6, 7, 19-22 and 24-29. Furthermore, although the Examiner has rejected Claims 8-11 as anticipated by Ebcioglu, the Examiner fails to show that each and every element of the claimed subject matter is disclosed in Ebcioglu, as required to establish a *prima facie* case of anticipation. Id.

According to the Examiner, Ebcioglu (col. 4, lines 22-26; col. 7, lines 13-16; col. 9, lines 51-63) teaches the bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value, as recited by Claim 8. (*See*, pg. 2, item 3c of the Final Office Action mailed November 5, 2004.) However, after careful review of the relevant portions of Ebcioglu cited by the Examiner, Applicant respectfully disagrees with the Examiner's contention.

Applicant respectfully submits that the Examiner has improperly equated the limiting of the amount of speculative instructions re-executed during speculative exception resolution (*see*, col. 4, lines 33-37) with the bypassing of execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value. In fact, Applicant respectfully submits that the claimed subject matter is directly contrary to the explicit teachings of Ebcioglu. As taught by Ebcioglu:

> where a speculative instruction attempts to use a register having its exception bit set . . . a secondary speculative exception is created. (col. 4, lines 13-16.)

As further taught by Ebcioglu:

> Speculative exception resolution is triggered when a non-speculative instruction, which is in the taken path, uses an operand from a register having its exception bit set. (col. 4, lines 26-29.)

According to the Examiner's above-described rationale, the bypassing of such instructions would prohibit both the detection of secondary speculative exceptions, as taught by Ebcioglu, as well as speculative exception resolution, as taught by Ebcioglu. Furthermore, Applicant respectfully submits that the limiting of re-execution of speculative instructions during speculative exception resolution, as taught by Ebcioglu (*see*, col. 4, lines 24-27) does not depend on whether such non-executed instructions have at least one operand with an associated tag that indicates that the operand is a scratch value. In other words, speculative instructions, which are not re-executed by Ebcioglu, are limited to speculative instructions that are either not dependent on the

instruction that caused the exception; or if such speculative instruction is dependent on the instruction that caused the exception, such speculative instruction is not re-executed if the speculative instruction is not in the taken path.

Hence, Applicant respectfully submits that the limiting of re-execution of speculative instructions during speculative exception resolution, as taught by Ebcioglu, fails to disclose each and every feature of the claimed subject matter. However, the case law is clear in establishing that each and every element of the claim must be exactly disclosed in the anticipatory reference. Id. Therefore, a *prima facie* case of anticipation of Claim 8 has not been established and the rejection of Claims 8-11 is therefore erroneous. Id.

2. Specific Limitations Not Described in the Art

Claim 8 recites:

> an execution engine to bypass execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value. (Emphasis added.)

3. Explanation Why Such Limitations Render the Claims Unanticipated By the Prior Art

Here, Ebcioglu fails to disclose an execution engine to bypass execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value, as recited by Claim 8. In contrast, Ebcioglu teaches that a secondary speculative exception is created when a speculative instruction attempts to use a register having its exception bit set. (*See*, col. 4, lines 13-16.) Furthermore, Ebcioglu teaches that a non-speculative instruction, which uses an operand from a register having its exception bit set, triggers speculative exception resolution if the non-speculative instruction is in the taken path. (*See*, col. 4, lines 26-29.)

Based on the cited passages above, Applicant respectfully submits that the entire text of Ebcioglu fails to disclose the bypassing of execution of instructions, which contain a scratch value in an operand. Applicant respectfully submits that such a teaching within Ebcioglu would be directly contrary to the explicit teachings of Ebcioglu and would, in fact, prohibit the detection of secondary speculative exceptions, as well as the triggering of speculative exception resolution. Accordingly, not only does Ebcioglu fail to disclose each and every element of the claimed subject matter; the teachings of Ebcioglu are directly contrary to the claimed subject matter, which bypasses instructions if a tag value of an operand indicates that the operand contains a scratch value.

However, the case law is clear in establishing that each and every element of the claim must be exactly disclosed by the anticipatory reference. Id. Therefore, Applicant respectfully submits that the failure of Ebcioglu to disclose each and every element recited by Claim 8 prohibits the Examiner's use of Ebcioglu as an anticipatory reference to establish a *prima facie* case of anticipation of Claim 8. Id.

Therefore, Applicant respectfully submits that a *prima facie* case of anticipation of Claims 8-11 is not established and the rejection of Claims 8-11 is erroneous and should be overturned. Accordingly, Applicant respectfully requests that the §102(b) rejection of Claims 8-11 be overturned.

D. Rejection of Claims 35-38 as Obvious Over Parady

The Examiner rejected pending Claims 35-38 under 35 U.S.C. §102(b) as being unpatentable over Parady.

1. Errors of Law and Fact in the Rejection

Although the Examiner has rejected Claims 35-38 as anticipated by Parady, the Examiner fails to show that each and every element of the claimed subject matter is exactly disclosed in Parady, as required to establish a *prima facie* case of anticipation under 35 U.S.C. §102(b). Banner Titanium, supra.

According to the Examiner, element 80 of FIG. 2 of Parady teaches the setting of a tag of a destination register in response to a cache miss to identify the register as containing a scratch value. FIG. 2 of Parady illustrates L2 cache tags, which are known to those skilled in the art to determine whether requested data is contained within an L2 cache. The only reference to element 80 shown in FIG. 1 of Parady is the description that L2 cache tag memory 80, and L2 cache data memory 82 are shown in FIG. 2. (*See*, col. 3, lines 27-28.)

In contrast to setting a tag register to identify the register as containing a scratch value in response to a cache miss, Parady is directed to a method and apparatus for switching between threads of a program in response to a long latency event. As described within Parady:

> The indication that a thread switch is required is provided on a line 114 providing an L2-miss indication from cache control/system interface 22 of FIG. 1. Upon such an indication, a switch to the next thread will be performed, using, in one embodiment, the next thread pointer on line 116. The next thread pointer is two bits indicating the next thread from an instruction which caused the cache miss. (col. 3, lines 58-65.) (Emphasis added.)

As further described by Parady:

> These two bits of the next thread pointer come from a thread field 118 in an instruction 120. Instruction 120 also includes an opcode field 122 and a source and destination register fields 124 and 126, respectively. By adding the two bit thread field 118 to appropriate instructions, control can be maintained over thread switching operations. In one embodiment, the thread field is added to all load and store operations. (col. 3, line 66 - col. 4, line 6.) (Emphasis added.)

Applicant respectfully submits that the Examiner has incorrectly equated a thread switch to bypassing execution of instructions, as recited by the claimed subject matter. As clearly illustrated by the cited passages above, the addition of a two-bit thread field 118 to appropriate

instructions provides control over thread switching operations, such as, for example, load and store instructions, which are indicated by Parady and recognized by those skilled in the art to represent long latency instructions.

Accordingly, once a thread switch occurs, the instruction that caused the thread switch will attempt to complete while a next thread begins execution. However, once a thread switch occurs, execution of instructions within the previous thread is paused while the instruction that initially caused the thread switch is completed. As explicitly described by Parady:

> In a preferred embodiment, upon completion of the memory access which caused the thread switch, the thread with the memory access must wait until it is pointed to again by the round robin or thread pointer bits to continue with its operation. Alternatively, a particular thread could be identified as a critical thread, and generate an interrupt as soon as the memory access is completed. (col. 4, lines 43-48.) (Emphasis added.)

Accordingly, based on the cited passage above, Applicant respectfully submits that the switching between threads in response to long latency events does not cause instructions to be bypassed since execution of such instructions will resume once the thread is pointed to again by the round robin, or thread pointer bits, to continue with its operation. (*See*, col. 4, lines 43-46.) Hence, Applicant submits that the entire specification of Parady is devoid of any reference to bypassing execution of instructions, which has at least one operand with an associated tag, but indicates that the operand is a scratch value.

The case law is clear in establishing that each and every element of the claim must be exactly disclosed in the anticipatory reference. Banner Titanium, supra. Hence, a *prima facie* case of anticipation of the claims by Parady has not been established and the rejection of Claims 35-38 is therefore erroneous.

2. Specific Limitations Not Described in the Prior Art

Claim 35 recites the following claim features

> setting a tag of a register when an instruction having the register as a destination results in a cache miss, to identify the register as containing a scratch value;
>
> bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value until at least one instruction is detected that results in a cache miss; and
>
> re-executing bypassed instructions once a cache is loaded with cache miss data. (Emphasis added.)

3. Explanation Why Such Limitations Render the Claims Non-Obvious Over the Prior Art

Applicant claims a system and method to bypass execution of instructions involving unreliable data during speculative execution. In contrast, the teachings of Parady are directed to multi-thread executions, specifically, the switching between threads in response to long latency events.

In contrast to the Applicant's claimed subject matter, Parady teaches the switch from a current thread to a next thread, in response to a long latency operation, such as a load or store operation. (*See*, col. 3, lines 58-65.) Applicant respectfully submits that a thread switch, as taught by Parady, does not result in the bypassing of execution of instructions, but instead pauses instruction execution within the current thread while instruction execution begins within a next thread. As explicitly recited by Parady:

> Upon completion of the memory access which caused the thread switch, the thread with the memory access must wait until it is pointed to again by the round robin, the thread pointer bits to continue with its operation. (col. 4, lines 42-46.) (Emphasis added.)

As indicated by the cited passage, the thread with the memory access, which caused the thread switch, must wait until it is pointed to again to continue operation. Until it is pointed to again, instructions are neither bypassed nor executed within that thread, as recited by the claimed subject matter. Specifically, instructions following the memory access, which caused the thread switch, are not bypassed until a cache miss is detected, but instead, execution of such instructions is paused until the thread is, once again, pointed to by the thread pointer bits. (*See*, supra.)

Furthermore, even assuming, arguendo, that Parady taught the bypassing of execution of instructions, Parady fails to disclose that the bypassing of instructions continues until at least one instruction is detected that results in a cache miss. In contrast, assuming that a thread switch teaches the bypassing of execution of instructions, resumption of execution of instructions following the latency causing instruction does not resume until the thread is pointed to again by the thread pointer bits. Hence, Parady fails to disclose bypassing of instructions until an instruction is detected that results in a cache miss.

Accordingly, Applicant respectfully submits that Parady fails to disclose the bypassing of execution of instructions, as well as the detection of an instruction, which results in a cache miss, to terminate the bypassing execution of such instructions. Therefore, since Parady fails to disclose bypassing execution of instructions, Parady also fails to disclose the re-execution of bypass instructions once the cache is loaded with cache miss data, as recited by Claim 35.

Yet, in spite of the lack of any teaching toward either the bypassing of execution of instructions until an instruction is detected that results in a cache miss, or the re-executing of such instructions, the Examiner incorrectly finds the disclosure of each and every element of the claimed

subject matter within Parady to anticipate Claims 35-38. However, the case law is clear in establishing that each and every element of the claim must be exactly disclosed in the anticipatory reference. Banner Titanium, supra.

Accordingly, Applicant respectfully submits that the Examiner fails to show that each and every element of the claimed subject matter is disclosed in Parady, as required to establish a *prima facie* case of anticipation under §102(b). Id. Therefore, a *prima facie* case of anticipation of the claims is not established, and the rejection of Claims 35-38 is erroneous and should be overturned. Accordingly, Applicant respectfully requests that the §102(b) rejection of Claims 35-38 be overturned.

E. Rejection of Claims 12-15 and 17 as Obvious Over Witt in View of Ebcioglu

The Examiner rejected pending Claims 12-15 and 17 under 35 U.S.C. §103(a) as being unpatentable over Witt in view of Ebcioglu.

1. Errors of Law and Fact in the Rejection

For the reasons provided below, the Examiner has failed to show that the prior art references of Witt in view of Ebcioglu teach or suggest all claim features of Claims 12-15, 17 and 18. The Federal Circuit Court of Appeals in In re Rijckaert, 9 F.3d 1531, 28 U.S.P.Q. 2d 1955 (Fed. Cir. 1993) held that:

> In rejecting claims under 35 U.S.C. § 103, the examiner bears the initial burden of presenting a *prima facie* case of obviousness. . . . "A *prima facie* case of obviousness is established when the teaching from the prior art itself would appear to have suggested the claimed subject matter to a person of ordinary skill in the art.". . . If the examiner fails to establish a *prima facie* case, the rejection is improper and will be overturned. (Emphasis added.) 9 F.3d at 1532, 28 U.S.P.Q. 2d at 1956.

Applicant respectfully submits that the combined teachings of Witt in view of Ebcioglu would not have suggested the claimed invention to one of ordinary skill in the art, as required to establish a *prima facie* case of obviousness. Id. Hence, a *prima facie* case of obviousness has not been established and the rejection is erroneous and should be overturned. Id.

According to the Examiner, Witt teaches

a. processor comprising at least two arithmetic units (FIG. 6B, 840R, 845R);
b. a translation look aside buffer (FIG. 6B, 915);
c. a branch prediction unit (FIG. 6A, element 825). (*See*, pg. 7, items 23a, 23b and 23c of the Final Office Action mailed November 5, 2004.)

Assuming, arguendo, that Witt teaches the above-described features of Claim 12 as correctly indicated by the Examiner, Witt fails to teach an execution engine having a plurality of instructions, which when executed, cause the processor to perform actions, including bypassing

execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value generated during speculative execution of a processor.

In contrast, Witt describes a high performance, superscalar microprocessor including a common reorder buffer and common register file for both integer and floating point units, which share a data processing bus (*See*, col. 10, lines 15-24), a common load store unit and a common branch unit on data processing bus (*See*, col. 10, lines 29-36). However, Witt is devoid of any teachings regarding the resolution of exceptions caused by speculative instructions.

As explicitly taught by Witt:

> When an exception occurs in the execution of a particular instruction, no global action is required. Rather, the exception status is merely reflected and the result's status returned to ROB 240. (col. 29, lines 62-65.)

As further described by Witt:

> When ROB 240 encounters the exception status in the process of retiring instructions, it initiates a drop operation, which consists of clearing all entries from ROB 240, asserting an exception signal to all functional units to clear them (and IDECODE) generating a trap vector per the VF bit and redirecting the sensor 257 to trap handling code. (col. 30, lines 6-12.)

In spite of the lack of any teachings or suggestions regarding the resolution of exceptions caused by speculative instructions, according to the Examiner:

> It would have been obvious to one of ordinary skill in the art at the time the invention was made to have the invention of Witt include the claimed execution engine and instructions, as taught by Ebcioglu, for the desirable purpose of reducing the overhead from exceptions caused by speculative instructions. (Ebcioglu, et al., col. 3, lines 58-61.) (*See*, pg. 9, item 26 of the Final Office Action mailed November 5, 2004.)

In other words, Applicant respectfully submits that the teachings from the prior art, itself, or Witt in view of Ebcioglu, fail to provide some suggestion or motivation for combining the reference teachings.

Assuming that one of skill in the art would recognize that speculative evaluation and execution, as defined by Ebcioglu, improved processor speed and efficiency, the Examiner fails to illustrate some rationale for combining the missing elements provided by Ebcioglu within the teachings of Witt. Accordingly, Applicant's claimed invention could only be arrived at through inappropriate hindsight.

It is well established that obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention absent the teaching or suggestion supporting such combination. ACS Hospital Sys., Inc. v. Montefiore Hospital, 732 F.2d. 1572, 1577, 221 U.S.P.Q. 929, 933 (Fed. Cir. 1984). Also, one cannot find obviousness through hindsight to construct a claimed invention from elements of the prior art. In re Warner, 379 F.2d 1011, 1016, 154 U.S.P.Q. 173, 177 (C.C.P.A. 1967).

Here, the Examiner has engaged in a prohibited hindsight based analysis to construct the claimed invention from elements of Witt and Ebcioglu due to the absence of any suggestion or motivation to combine the teaching of Witt in view of Ebcioglu. Id. Furthermore, Applicant respectfully submits that the combined teachings of Witt and Ebcioglu would not have suggested the claimed subject matter to one of ordinary skill in the art, as required to establish a *prima facie* case of obviousness. In re Rijckaert, *supra*. Hence, a *prima facie* case of obviousness has not been established and the rejection is erroneous and should be overturned.

2. Specific Limitations Not Described in the Prior Art

Claim 12 recites the following claim features:

> at least two arithmetic units;
> a translation look aside buffer;
> a branch prediction unit; and
> an execution engine having a plurality of instructions which when executed cause the processor to perform actions including:
> identifying scratch values generated during speculative execution of a processor,
> setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and
> bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value. (Emphasis added.)

3. Explanation Why Such Limitations Render the Claims Non-Obvious Over the Prior Art

The Examiner fails to illustrate that the combination or modification of Witt in view of Ebcioglu teaches or suggests each of the recited features of the claimed invention. However, the case law is clear in establishing that "to establish *prima facie* obviousness of a claimed invention, all of the claim limitations must be taught or suggested by the prior art." In re Royka, 490 F.2d 981, 180 U.S.P.Q. 580 (CCPA 1974).

Here, the claimed invention recites:

> an execution engine to bypass execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value generated during speculative execution of a processor. (Emphasis added.)

As indicated above, limiting an amount of speculative instructions re-executed during speculative exception resolution, as taught by Ebcioglu, fails to teach or suggest the bypassing of execution of instructions, as recited by Claim 12. (*See*, Ebcioglu, col. 3, lines 62 through col. 4, line 1.) Furthermore, as indicated above, even assuming, arguendo, that prohibiting re-execution of speculative instructions, which are either not dependent on a speculative instruction, which caused the exception, or if dependent on such an instruction not in the taken path, such a

determination is not based on whether the instruction includes an operand with an associated tag that indicates that the operand is a scratch value generated during speculative execution of a processor.

In contrast, Applicant respectfully submits that Ebcioglu teaches away from the above-recited features of Claim 12. Specifically, as taught by Ebcioglu, an instruction which uses an operand from a register having its exception bit set will either trigger a speculative exception resolution, assuming the instruction is in a taken path and non-speculative, or create a secondary exception, assuming the instruction is speculative. (*See*, col. 4, lines 13-29.)

Applicant respectfully submits that the bypassing of such instructions would prohibit the triggering of a speculative exception resolution, as taught by Ebcioglu, as well as secondary speculative exceptions, as taught by Ebcioglu. Accordingly, Applicant respectfully submits that the Examiner fails to establish a *prima facie* case of obviousness of the claimed invention since the Examiner fails to illustrate the teaching or suggestion of all claim features of Claim 12 within the prior art. Id.

Furthermore, Applicant respectfully submits that the Examiner has engaged in a hindsight-based analysis to provide some rationale for combining Ebcioglu with the teachings of Witt. As a result, Applicant respectfully submits that the Examiner fails to establish that it would be obvious to combine the teachings of Ebcioglu with the teachings of Witt. In re Warner, *supra*.

Accordingly, Applicant respectfully submits that the combined teachings of Witt and Ebcioglu would not have suggested the claimed invention to one of ordinary skill in the art as is required to establish a *prima facie* case of obviousness. In re Rijckaert, *supra*. Hence, a *prima facie* case of obviousness has not been established and the rejection is erroneous and should be overturned. Id. Accordingly, Applicant respectfully requests that the §103(a) rejection of Claims 12-15 and 17 be overturned.

F. Rejection of Claims 7 and 25 as Anticipated by Ebcioglu

The Examiner rejected pending Claims 7 and 25 under 35 U.S.C. §102(b) as being anticipated by Ebcioglu.

1. Errors of Law and Fact in the Rejection

The Examiner has made the same errors as previously described with respect to the rejection of Claims 1-4, 6, 19-22, 24 and 26-29. Furthermore, although the Examiner has rejected Claims 7 and 25 as anticipated by Ebcioglu, the Examiner fails to show that each and every element of Claims 7 and 25 is disclosed in Ebcioglu, as required to establish a *prima facie* case of anticipation. Id.

According to the Examiner, Ebcioglu (col. 5, lines 30-44, the 33rd bit.) teaches propagating the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value, as recited by Claims 7 and 25. (*See*, pg. 3, item 8 of Final Office Action mailed November 5, 2004.) However, after careful review of the relevant portions of Ebcioglu cited by the Examiner, Applicant respectfully disagrees with the Examiner's contention.

Applicant respectfully submits that the introduction of an exception bit within registers 102, as shown in FIG. 1A, as well as the introduction of the extra bit to VLIW instructions 106, as shown in FIG. 1B of Ebcioglu (*see*, col. 5, lines 30-44) fails to disclose the propagating of a tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value, as recited by Claims 7 and 25.

Applicant respectfully submits that the above-cited passage of Ebcioglu, is explicitly limited to disclosing the addition of a speculative bit to VLIW instructions (*see*, col. 5, lines 39-43) and the addition of a speculative bit to registers within a VLIW machine (*see*, col. 5, lines 31-35.) Hence, Ebcioglu fails to provide any disclosure regarding the propagation of a tag value through a store buffer based on whether an address generation register holds a scratch value, as recited by Claims 7 and 25.

Hence, Applicant respectfully submits that the addition of a speculative bit to a register and to an instruction, as taught by Ebcioglu, fails to disclose each and every feature of the claimed subject matter. However, the case law is clear in establishing that each and every element of the claim must be exactly disclosed in the anticipatory reference. Id. Therefore, a *prima facie* case of anticipation of Claims 7 and 25 has not been established and the rejection of Claims 7 and 25 is therefore erroneous. Id.

2. Specific Limitations Not Described in the Prior Art

Claims 7 and 25 recite:

propagating the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value.

3. Explanation Why Such Limitations Render the Claims Unanticipated by the Prior Art

Here, Ebcioglu fails to disclose the propagation of a tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value, as recited by Claims 7 and 25. The features recited by Claims 7 and 25 refer to a class of instructions that, although marked as scratch values, include address generation operands that are not marked as scratch values. Hence, Claims 7 and 25 recite that, for this class of instructions, the SVal tag is propagated to the store buffer. (*See*, pg. 9, ¶0026 of Applicant's specification.)

Accordingly, Applicant respectfully submits that col. 5, lines 30-44 of Ebcioglu, as cited by the Examiner, as well as the entire specification of Ebcioglu, fail to provide any teachings or suggestions with regards to the above-recited features of Claims 7 and 25. However, the case law is clear in establishing that each and every element of the claim must be exactly disclosed by the anticipatory reference. Id.

Therefore, Applicant respectfully submits that the failure of Ebcioglu to disclose each and every element recited by Claims 7 and 25, prohibits the Examiner's use of Ebcioglu as an anticipatory reference to establish a *prima facie* case of anticipation of Claims 7 and 25. Id.

Consequently, Applicant respectfully submits that a *prima facie* case of anticipation of Claims 7 and 25 is not established and the rejection of Claims 7 and 25 is erroneous and should be overturned. Accordingly, Applicant respectfully requests that the §102(b) rejection of Claims 7 and 25 be overturned.

G. Rejection of Claim 18 as Obvious Over Witt in View of Ebcioglu

The Examiner rejected pending Claim 18 under 35 U.S.C. §103(a) as being obvious over Witt in view of Ebcioglu.

1. Errors of Law and Fact in the Rejection

The Examiner has made the same errors as previously described with respect to the rejection of Claims 7, 12-15, 17 and 25. Applicant respectfully submits that the recited features of Claim 18 are analogous to the recited features of Claims 7 and 25. Accordingly, the arguments regarding Claims 7 and 25 apply to the Examiner's rejection of Claim 18 as obvious over Ebcioglu in view of Witt.

For at least the reason provide above regarding the rejection of Claims 7 and 25, Applicant respectfully submits that the combined teachings of Witt in view of Ebcioglu would not have suggested the claimed subject matter to one of ordinary skill in the art, as required to establish a *prima facie* case of obviousness. In re Rijckaert, supra.

As described above, Witt is devoid of any teachings regarding the resolution of exceptions caused by speculative instructions. Therefore Applicant respectfully submits that since Witt is devoid of any teachings or suggestions with regard to speculative exception processing, the Examiner is limited to illustrating a teaching or suggestion within Ebcioglu to render the above-recited features of Claim 18 obvious.

Regarding Ebcioglu, as indicated above with regard to the rejection of Claims 7 and 25, the addition of an exception bit to registers or instructions, as taught by Ebcioglu, fails to teach or suggest propagating of the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value, as recited by Claim 18.

Hence, for at least the reasons indicated above, the Examiner is prohibited from establishing that all of the claim limitations are taught or suggested by the prior art of Witt in view of Ebcioglu, as required to establish a *prima facie* case of obviousness. In re Royka, supra.

Consequently, Applicant respectfully submits that the Examiner fails to establish a *prima facie* case of obviousness of the claimed invention, since the Examiner fails to illustrate the teaching or suggestion of all claim features of Claim 18 within the prior art. Id.

Accordingly, Applicant respectfully submits that the combined teachings of Witt and Ebcioglu would not have suggested the claimed subject matter to one of ordinary skill in the art, as required to establish a *prima facie* case of obviousness. In re Rijckaert, supra. Hence, a *prima facie* case of obviousness has not been established and the rejection of Claim 18 is erroneous and should be overturned. Therefore, Applicant respectfully requests that the §103(a) rejection of Claim 18 be overturned.

2. Specific Limitations Not Described in the Prior Art

Claim 18 recites the following claim features:

propagating the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value.

3. Explanation Why Such Limitations Render the Claims Non-Obvious Over the Prior Art

Applicant respectfully submits that the recited features of Claim 18 are analogous to the recited features of Claims 7 and 25. Accordingly, the arguments regarding Claims 7 and 25 apply to the Examiner's rejection of Claim 18 as obvious over Ebcioglu in view of Witt.

As indicated above, the Examiner is limited to illustrating a teaching or suggestion within Ebcioglu to render the above-recited features of Claim 18 obvious. However, for at least the reasons indicated above, the Examiner is prohibited from establishing that all of the claim limitations are taught or suggested by the prior art of Witt in view of Ebcioglu, as required to establish a *prima facie* case of obviousness. In re Royka, supra.

Accordingly, Applicant respectfully submits that the combined teachings of Witt and Ebcioglu would not have suggested the claimed subject matter to one of ordinary skill in the art, as required to establish a *prima facie* case of obviousness. In re Rijckaert, supra. Hence, a *prima facie* case of obviousness has not been established and the rejection of Claim 18 is erroneous and should be overturned. Therefore, Applicant respectfully requests that the §103(a) rejection of Claim 18 be overturned.

Based on the foregoing, Applicant requests that the Board overturn the rejection of all pending claims and hold that all of the claims of the present application are allowable.

Respectfully submitted,

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN

Dated: July 12, 2005

By: _____
Joseph Lutz, Reg. No. 43,765

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, California 90025
(310) 207-3800

CERTIFICATE OF MAILING:
I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, with sufficient postage, in an envelope addressed to: Mail Stop Appeal Brief - Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on July 12, 2005

_____
Marilyn Bass July 12, 2005

VIII. CLAIMS APPENDIX

The claims involved in this Appeal are as follows:

1. A method comprising:

identifying scratch values generated during speculative execution of a processor;

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and

bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value.

2. The method of claim 1 wherein setting comprises setting the tag of a register when an instruction having the register as a destination results in a cache miss.

3. The method of claim 1 further comprising:

propagating the tag to destination registers of the instructions to indicate that an operand within the destination registers is a scratch value.

4. The method of claim 1 further comprising:

bypassing execution of an arithmetic instruction having at least one register as an operand with an associated tag indicating that the register contains data that is a scratch value; and

bypassing execution of a store instruction involving a value derived from a register having an associated tag indicating that the register contains data that is a scratch value.

5. The method of claim 1 further comprising:

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

6. The method of claim 1 further comprising:

marking each instruction in a pipeline with a tag to indicate if the instruction involves a scratch value.

7. The method of claim 1 further comprising:

propagating the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value.

8. A processor comprising:

a plurality of registers having a corresponding plurality of register tags to indicate whether the data stored in the register holds a scratch value;

a plurality of flags having a corresponding plurality of flag tags to indicate whether the data reflected by the flag is based on a scratch value;

a plurality of predicates having a corresponding plurality of predicate tags to indicate whether the data reflected by the predicate is based on a scratch value; and

an execution engine to bypass execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value.

9. The processor of claim 8 having an instruction set including a plurality of instructions, each instruction augmented by an instruction tag to indicate whether the instruction involves a scratch value.

10. The processor of claim 9 wherein the register tags, flag tags, predicate tags, and instruction tags have a size of one bit.

11. The processor of claim 9 wherein the register tags, flag tags, predicate tags and instruction tags have a size of at least two bits.

12. A processor comprising:

at least two arithmetic units;

a translation look aside buffer;

a branch prediction unit; and

an execution engine having a plurality of instructions which when executed cause the processor to perform actions including:

identifying scratch values generated during speculative execution of a processor,

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and

bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value.

13. The processor of claim 12 wherein setting comprises setting the tag of a register when an instruction having the register as a destination results in a cache miss.

14. The processor of claim 12 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

propagating the tag to destination registers of the instructions to indicate that the operand within the destination registers is a scratch value.

15. The processor of claim 12 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

bypassing execution of an arithmetic instruction having at least one register as an operand with an associated tag indicating that the register contains data that is a scratch value; and

bypassing execution of a store instruction involving a value derived from the register having an associated tag indicating that the register contains data that is a scratch value.

16. The processor of claim 12 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

17. The processor of claim 12 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

marking each instruction in a pipeline with a tag to indicate if the instruction involves a scratch value.

18. The processor of claim 12 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

propagating the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value.

19. A system comprising:

a memory, a storage device, and a processor each coupled to a bus;

the processor including an execution engine having instructions which when executed by the processor cause the processor to perform actions including:

identifying scratch values generated during speculative execution of a processor;

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and

bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value.

20. The system of claim 19 wherein setting comprises setting the tag of a register when an instruction having the register as a destination results in a cache miss.

21. The system of claim 19 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

propagating the tag to destination registers of the instructions indicate that an operand within the destination registers is a scratch value.

22. The system of claim 19 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

bypassing execution of an arithmetic instruction having at least one register as an operand with an associated tag indicating that the register contains data that is a scratch value; and

bypassing execution of a store instruction involving a value derived from the register having an associated tag indicating that the register contains data that is a scratch value.

23. The system of claim 19 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

24. The system of claim 19 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

marking each instruction in a pipeline with a tag to indicate if the instruction involves a scratch value.

25. The system of claim 19 wherein the execution engine has further instructions which when executed cause the processor to perform further actions comprising:

propagating the tag through a store buffer if an address generation register does not indicate that the address generation register holds a scratch value.

26. A machine readable medium having instructions stored thereon which when executed by a processor cause the processor to perform actions including:

identifying scratch values generated during speculative execution of a processor;

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and

bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value.

27. The machine readable medium of claim 26 wherein setting comprises setting the tag of a register when an instruction having the register as a destination results in a cache miss.

28. The machine readable medium of claim 26 having further instructions which when executed cause the processor to perform further actions comprising:

propagating the tag to destination registers of the instructions to indicate that an operand within the destination registers is a scratch value.

29. The machine readable medium of claim 26 having further instructions which when executed cause the processor to perform further actions comprising:

bypassing execution of an arithmetic instruction having at least one register as an operand with an associated tag indicating that the register contains data that is a scratch value; and

bypassing execution of a store instruction involving a value derived from the register having an associated tag indicating that the register contains data that is a scratch value.

30. The machine readable medium of claim 26 having further instructions which when executed cause the processor to perform further actions comprising:

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

31. A method comprising:

identifying scratch values generated during speculative execution of a processor;

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

32. A processor comprising:

at least two arithmetic units;

a translation look aside buffer;

a branch prediction unit; and

an execution engine having a plurality of instructions which when executed cause the processor to perform actions including:

identifying scratch values generated during speculative execution of a processor,

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value, and

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

33. A system comprising:

a memory, a storage device, and a processor each coupled to a bus;

the processor including an execution engine having instructions which when executed by the processor cause the processor to perform actions including:

identifying scratch values generated during speculative execution of a processor,

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value, and

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

34. A machine readable medium having instructions stored thereon which when executed by a processor cause the processor to perform actions including:

identifying scratch values generated during speculative execution of a processor;

setting at least one tag associated with at least one data area of the processor to indicate that the data area holds a scratch value; and

utilizing a branch predictor to override computed branch results produced by branch instructions based on data having a tag that indicates the data is a scratch value.

35. A method comprising:

setting a tag of a register when an instruction having the register as a destination results in a cache miss, to identify the register as containing a scratch value;

bypassing execution of instructions having at least one operand with an associated tag that indicates that the operand is a scratch value until at least one instruction is detected that results in a cache miss; and

re-executing bypassed instructions once a cache is loaded with cache miss data.

36. The method of claim 35, further comprising:

executing store instructions having a tag to indicate that the instruction involves a scratch value if tag values associated with operands of the store instruction indicate non-scratch values.

37. The method of claim 35, further comprising:

propagating the tag to destination registers of the instructions to indicate that an operand within the destination registers is a scratch value.

38. The method of claim 35, further comprising:

servicing detected cache miss instructions in parallel prior to re-executing the bypassed instructions.